集合覆盖问题的数据约简研究 *

陈 韬, 吴志勇, 孙乐昌, 张 , 刘京菊
(解放军电子工程学院 合肥 230037)

摘 要: 针对当前解决大规模集合覆盖问题的算法普遍存在着效率不高的问题,提出了一套削减数据规模的约简方法,并给出了一个能够与其他所有解决集合覆盖问题算法相结合的约简算法。用 Beasley 提出的 45 个测试用例进行试验,结果显示贪心算法和遗传算法在结合了约简算法后能够在更少的时间内得到更优的解,表明该约简方法和约简算法可以有效提高传统算法和智能算法解决大规模集合覆盖问题的效率。

关键词: 集合覆盖; 约简方法; 约简算法; 贪心算法; 遗传算法
中图分类号: TP391 **文献标志码**: A **文章编号**: 1001-3695(2010)09-3307-05
doi:10.3969/j.issn.1001-3695.2010.09.028

Research of data reduction on set covering problem

CHEN Tao, WU Zhi-yong, SUN Le-chang, ZHANG Ming, LIU Jing-ju
(*Electronic Engineering Institute of PLA*, *Hefei* 230037, *China*)

Abstract: When solving the large-scale set covering problem (SCP), there is a universal efficiency problem for most algorithms. To solve the problem, this paper proposed a suite of reduction theories to reduce the data scale and gave a universal DRA which could be combined with all the algorithms solving SCP. 45 instances proposed by Beasley are tested. Experiments results show that after combined with DRA, the greedy algorithm and genetic algorithm can achieve better solutions with less time, which indicates that the reduction theory and DRA can effectively improve the efficiency for both traditional algorithms and intelligent algorithms to solve SCP.

Key words: set covering; reduction theory; data reduction algorithm(DRA); greedy algorithm; genetic algorithm

集合覆盖问题(SCP)是运筹学中最基本的组合问题。它已被广泛应用于机组人员调度[1~4]、开关电路设计[5]、生产线最佳编制[6]、货车递送[7]、市场交易分析[8]和情报检索[9]等。SCP 问题已被证明为 NP 完全问题[10]。

自从集合覆盖问题提出以来,相继有很多学者利用不同思想提出了很多不同的算法,这些算法主要分为完整算法[11]和启发式算法。完整算法基本上建立在分支定界基础上。通过比较和分析,Caprara 等人[12]认为 CPLEX 算法是求解 SCP 最好的完整算法。但如果问题规模比较大时,其时间代价会非常高。而启发式算法则以牺牲解的精度来取得较好的时间复杂度,在可接受的时间内找到一个最优近似解。在实际问题中,最优近似解一般也能够满足现实的要求。公认比较好的启发式算法有贪心算法[13]、遗传算法[14~16]、模拟退火算法[17]、蚁群算法[18]、拉格朗日算法[19,20]、人工神经网络算法[21]等。

原始数据规模对算法的执行效率有着直接的影响,而现实中大规模数据的问题经常出现,尽管一些启发式算法能取得相对较好的时间复杂度,但对于绝大多数算法仍然存在着效率不高的问题。显然,缩减数据的规模无论对于完整算法还是启发式算法,均可以缩小最优解或近似最优解的搜索空间,从而提高解决集合覆盖问题的效率。为此本文在给出的一系列约简方法的基础上,提出了一个通用的数据约简算法,即在不影响最优解或近似最优解的获取的前提下,尽可能缩小数据的规模。然后用 Beasley 提出的 45 个 SCP 实例对结合了该约简算法的贪心算法和遗传算法进行测试,实验结果表明该算法对于缩减 SCP 问题的规模是可行的。

1 问题描述

集合覆盖问题可用二值整数规划描述为

$$\text{minimize} \quad \sum_{j=1}^{n} c_j x_j \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} x_j \geqslant 1; i=1,2,\cdots,m \tag{2}$$

$$x_j \in (0,1); j=1,2,\cdots,n \tag{3}$$

式(1)中 c_j 表示选择 x_j 的代价;$x_j=1$ 为第 j 列被选中,$x_j=0$则表示第 j 列未包含于解中;式(2)中 a_{ij}表示矩阵第 i 行、第 j 列的值,$a_{ij}=1$ 表示第 j 列覆盖了第 i 行,$a_{ij}=0$ 表示第 j 列没有覆盖第 i 行;表达式(2)表示每一行至少被一列覆盖。

表 1 为一个集合覆盖的实例。

在某一个时刻的所有属性构成一个状态,称为一个状态集(state)。一个状态集包括:有效行(effective_row),指当前状态下还未被覆盖的行,初始化状态下,有效行包括所有行;有效列(effective_column),指当前状态下还未被选择或去除的列,初始化状态下,有效列包括所有列;必选列(necessary_column),

收稿日期: 2010-02-07; **修回日期**: 2010-03-10 **基金项目**: 国家自然科学基金资助项目(60972161);电子工程学院博士生创新基金资助项目(CX2007016)

作者简介:陈韬(1987-),男,湖南益阳人,硕士研究生,主要研究方向为网络安全(oatnehc0924@126.com);吴志勇(1982-),男,博士研究生,主要研究方向为网络安全、软件测试;孙乐昌(1951-),男,教授,主要研究方向为网络安全、操作系统;张 (1966-)男,教授,主要研究方向为通信信号处理、模式识别、智能计算;刘京菊(1974-),女,讲师,硕士,主要研究方向为网络安全.

指一定包含在最优解中的列}。

表 1　集合覆盖实例

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
A	j_1	j_2	j_3	j_4	j_5	j_6	j_7	j_8
i_1	1	1	0	0	1	1	1	1
i_2	0	0	0	1	0	0	0	0
i_3	1	0	0	0	1	0	1	0
i_4	0	1	1	0	0	1	1	1
i_5	0	0	0	1	1	1	0	1
i_6	1	0	1	0	0	0	0	0
price	1	1	2	2	4	5	6	6

2　相关定义、定理及证明

定义 1　列包。矩阵的列 $j_p, p \in \{1,2,\cdots,n\}$ 所覆盖的行集 $I = \{i_q, q \in \{1,2,\cdots,m\}\}$，矩阵的列集 $J = \{j_{p'}, p' \in \{1,2,\cdots,n\}\}$ 所覆盖的行集为 $I' = \{i_{q'}, q' \in \{1,2,\cdots,m\}\}$，$J$ 不包含 j_p，如果存在 $I \subseteq I'$ 那么称 J 是 j_p 的列包。

实例中 $\{j_1, j_2\}$ 是 j_3 的列包；$\{j_2, j_4\}$、$\{j_1, j_3, j_4\}$ 是 j_6 的列包；$\{j_1, j_2, j_4\}$ 是 j_7 和 j_8 的列包…… 可以看出，同一列可以有多个不同列包，不同的列可以有相同的列包。

定义 2　冗余列。令列集 J 的代价为

$$\mathrm{cost}(J) = \sum_{j \in J} \mathrm{price}(j) \tag{4}$$

如果存在 j_p 的列包 J，使得：

$$\mathrm{price}(j_p) > \mathrm{cost}(J) \tag{5}$$

那么称 j_p 为冗余列(redundant_column)。

定义 3　最小列包。假设 j_p 有多个列包，分别为 $J_1, J_2, \cdots, J_t$。从这些列包中选出 J_0，使得 $\mathrm{cost}(J_0) = \min(\mathrm{cost}(J_1), \mathrm{cost}(J_2), \cdots, \mathrm{Cost}(J_t))$，称 J_0 为最小列包。

定义 4　行包。令当前状态下覆盖有效行 $i_k (k = 1,2,\cdots,n)$ 的有效列集为 J，覆盖行 $i_{k'} (k' \neq k, k' = 1,2,\cdots,n)$ 的有效列集为 J'，如果 $J \subseteq J'$，称 i_k 为 $i_{k'}$ 的行包。

特别地，若 J 的有效列数为 0，即没有列覆盖 i_k，称 i_k 为零覆盖行(zero_covered_row)；若有效列数为 1，表示只有一列 j 覆盖 i_k，称 i_k 为单覆盖行，列 j 为行 i_k 的覆盖列。

在实例中，i_2 为单覆盖行，j_4 是 i_2 的覆盖列；i_2 是 i_5 的行包；i_3 是 i_1 的行包。

定义 5　冗余行。当前状态下，如果行 i_k 是行 $i_{k'}$ 的行包，要使 i_k 被覆盖，一定要从 J 中选出某一列 j 覆盖 i_k，而根据行包的定义有 $J \in J'$，所以 $j \in J'$，即行 $i_{k'}$ 也一定被覆盖。因此如果当 i_k 被覆盖，$i_{k'}$ 一定会被覆盖时，称 $i_{k'}$ 为冗余行(redundant_row)。特别规定零覆盖行也为冗余行。

定理 1　如果 j_p 为冗余列，其最小列包为 J_0，那么 $\mathrm{price}(j_p) > \mathrm{cost}(J_0)$；反之也成立。

证明　j_p 为冗余列，根据定义 2，必定存在一个 j_p 的列包 J 使得 $\mathrm{price}(j_p) > \mathrm{cost}(J)$，而由最小列包定义可知 $\mathrm{cost}(J) \geqslant \mathrm{cost}(J_0)$，所以 $\mathrm{price}(j_p) > \mathrm{cost}(J_0)$；反之，如果 $\mathrm{price}(j_p) > \mathrm{cost}(J_0)$，显然列 j_p 符合冗余列的定义。

定理 2　冗余列一定不在最优解中。

证明　j_p 为冗余列，假设 j_p 在最优解 $J_{\mathrm{opt}'}$ 中，且该最优解代价为 $\mathrm{cost}_{\mathrm{opt}'}$，根据冗余列的定义，$j_p$ 最小列包为 J_0，那么解 $J_0 \cup (J_{\mathrm{opt}'} - j_p)$ 的代价应该为

$$\mathrm{cost}_{\mathrm{opt}'} - \mathrm{cost}(j_p) + \mathrm{cost}(J_0) < \mathrm{cost}_{\mathrm{opt}'} \tag{6}$$

说明解 $J_{\mathrm{opt}'}$ 的代价 $\mathrm{Cost}_{\mathrm{opt}'}$ 并不是最小代价，那么 $J_{\mathrm{opt}'}$ 不是最优解，这与假设相矛盾，即证。

定理 3　单覆盖行的覆盖列一定在最优解中。

证明　假设行 i_k 为单覆盖行，令其单覆盖列为 j，由于只有 j 能覆盖行 i_k，那么一个解要想覆盖行 i_k，必须包含 j，即单覆盖行的覆盖列是必选列，一定在最优解中。

定理 4　零覆盖行不能被任何解覆盖。

证明　若存在零覆盖行 i，由于初始状态下没有有效列覆盖 i，在算法的任何状态下覆盖 i 的有效列为 0。零覆盖行无法被任何解覆盖。

在二值整数规划中，行相当于一个约束表达式。根据前述的冗余行和冗余列的定义，冗余行就相当于一个没有约束的约束条件，它的去除并不影响其余行的覆盖。而冗余列的去除相当于标定该列的值为 0，对其他列的选择也没有影响，又根据定理 2，冗余列一定不在最优解中，因此可以将冗余行、冗余列从原始数据中去除而不影响其余行的覆盖和列的选择，并且不影响最优解的获取。将以上提到的定义、定理统称为约简方法。

3　算法分析及实验

3.1　算法描述

约简算法的设计以约简方法为依据。下面对约简算法作整体描述，并对其通用性和可行性进行分析。

3.1.1　约简算法

算法 1　约简算法

```
input: A, price
output: B, price, effective_column
        effective_row, zero_covered_row
1tag the zero_covered_row;//标记并去除零覆盖行
2last_state = null;
3while this_state ≠ last_state
4   last_state = this_state;//保存状态
5   delete null_column; //去除空列
6   delete redundant_column;//去除冗余列
7   select necessary_column; //选择并处理必选列
8build_new_matrix; //构造新矩阵并输出
```

3.1.2　算法通用性说明

为了使约简后的数据保持原始数据的格式，算法在最后构造了新矩阵，新矩阵由当前状态的有效行和有效列组成。

算法 2　生成新矩阵(build_new_matrix)

```
1 for i = 1: number_of_effective_row
2 for j = 1: number_of_effective_column
3 B(i,j) = A(effective_row(i), effective_column(j));
4 price(j) = Price(effective_column(j));
```

其中：effective_row(i)表示在矩阵 A 中第 i 个有效行的位置，effective_column(j)表示在矩阵 A 中第 j 个有效列的位置；新矩阵 B 大小为 $i \times j$。在后续集合覆盖算法中，直接对矩阵 B 和 price 进行操作，只要将得到的最优近似解的行列用 effective_row 和 effective_column 转换为 A 中的行列号，再结合 necessary_column，即得到矩阵的最优近似解。因此该算法基本上可以与任何解决 SCP 问题的算法相结合，即其通用性非常强。

3.1.3　算法可行性

根据以上算法描述，原始矩阵 A 转换成了矩阵 B 和一个必选列集 necessary_column。如图 1 所示：令 SA 为原始解集，

是由矩阵 A 中的列组成的全部可行解的集合;SB 为约简解集,每个解由矩阵 B 中的列和 necessary_column 组成,是去除了部分冗余列的所有可行解的集合;SO 表示最优解集,令最优解的代价为 cost_{opt}。

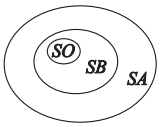

图1　解集关系

一个可行解 s 的代价为

$$\text{cost}(s) = \sum_{j \in s} \text{price}(j) \tag{7}$$

$\text{cost}(s)$ 越小,表示 s 越接近于最优解,称 s 的优化度越高。当 $\text{cost}(s) = \text{cost}_{\text{opt}}$ 时,s 为最优解。

一组可行解集合 S 的代价 $\text{cost}(S)$ 为

$$\text{cost}(S) = \sum_{s \in S} \text{cost}(s) \tag{8}$$

定义 6　期望代价。S 的期望代价 $E(S)$ 表示从 S 中搜索得到解的代价的期望值,定义为

$$E(S) = \text{cost}(S)/|S| \tag{9}$$

S 的期望代价越小,表示在同等条件搜索下,其搜索到的解的优化度越高。当 $E(S) = \text{cost}_{\text{opt}}$ 时,$E(S)$ 为最优解集。

定理 5　最优解一定在 SB 中,并且 SB 的优化度要高于 SA 的优化度。

证明　取任意 $s_a \in SA - SB$,根据上述定义,组成 s_a 的列中至少包含了一个冗余列,根据冗余列定义,必定能找到一个相对应的解 $s_b \in SB$,使得 $\text{cost}(s_b) < \text{cost}(s_a)$,同时,可以看出不同的 s_a 可能对应相同的 s_b,即从 $SA - SB$ 到 SB 的转换是多对一的关系,所以 $|SA - SB| > |SB|$。

显然　$E(SB) < E(SA - SB)$　(10)

因此 $SA - SB$ 中的解必定不是最优解,最优解一定在 SB 中。

综上所述,得:

$$E(SA) = \frac{|SA - SB|E(SA - SB) + |SB|E(SB)}{|SA|} > \tag{11}$$

$$\frac{|SA - SB|E(SB) + |SB|E(SB)}{|SA - SB| + |SB|} = \tag{12}$$

$$E(SB) \tag{13}$$

即 SB 的优化度要高于 SA 的优化度。

需要说明的是,SB 的优化度比 SA 的高,并不代表 SB 中任意一个解都比 SA 中的解好,而只是在同等条件的搜索下,从 SB 中得到高质量解的概率比 SA 大。通过算法的描述分析得出所有算法都是对原始数据的线性运算,因此时间复杂度为 $O(m \times n)$,该运算时间是可接受的。

结论　在有限的时间内经过约简算法转换,得到的数据 SB 的优化度要高于原始数据 SA,并且最优解仍然在 SB 中。本文算法对 SCP 数据的约简是可行的。

3.2　测试结果

利用本文提出的算法对 45 个 SCP 测试用例[22]进行约简,其实验平台为:CPU:Pentium(R)D 3.4 GHz;内存:2.98 GB;OS:Windows XP Professional 2002 + SP2。表 2 所示为约简前后测试用例的规模,T_p 表示约简算法运行的时间。为了更直观地对约简前后的规模进行比较,定义约简规模比为

$$1 - \frac{B_{\text{row}} \times B_{\text{column}}}{A_{\text{row}} \times A_{\text{column}}} \times 100\% \tag{14}$$

约简规模比越大,表示经过约简后数据的规模越小,数据优化度就越高。例如表中 SCP41 约简规模比为 95.30%,即表示 B 的规模相对于 A 的规模约简了 95.30%。

表 2　约简前后测试用例规模表

测试用例	A		B		必选列数	T_p/s	约简规模比/%
	row	column	row	column			
SCP41	200	1 000	99	95	28	0.17	95.30
SCP42	200	1 000	193	200	2	0.15	80.07
SCP43	200	1 000	192	209	2	0.16	80.00
SCP44	200	1 000	184	180	4	0.15	83.44
SCP45	200	1 000	153	159	10	0.17	87.84
SCP46	200	1 000	179	195	4	0.20	82.54
SCP47	200	1 000	171	156	7	0.17	86.66
SCP48	200	1 000	191	183	4	0.16	82.52
SCP49	200	1 000	195	218	1	0.22	78.74
SCP410	200	1 000	137	137	16	0.18	90.61
SCP51	200	2 000	198	200	1	0.30	90.10
SCP52	200	2 000	195	234	2	0.32	88.59
SCP53	200	2 000	195	201	1	0.29	90.20
SCP54	200	2 000	186	207	3	0.31	90.37
SCP55	200	2 000	120	128	16	0.30	96.16
SCP56	200	2 000	174	182	7	0.30	92.09
SCP57	200	2 000	174	189	6	0.35	91.77
SCP58	200	2 000	196	229	1	0.31	88.77
SCP59	200	2 000	168	182	7	0.33	92.36
SCP510	200	2 000	170	189	7	0.33	91.96
SCP61	200	1 000	200	184	0	0.19	81.6
SCP62	200	1 000	200	213	0	0.23	78.70
SCP63	200	1 000	200	204	0	0.23	79.60
SCP64	200	1 000	200	169	0	0.23	83.10
SCP65	200	1 000	192	200	1	0.24	80.80
SCPa1	300	3 000	300	344	0	0.69	88.53
SCPa2	300	3 000	300	370	0	0.70	87.66
SCPa3	300	3 000	300	363	0	0.67	87.90
SCPa4	300	3 000	300	355	0	0.64	88.16
SCPa5	300	3 000	300	363	0	0.74	87.90
SCPb1	300	3 000	300	367	0	0.91	87.76
SCPb2	300	3 000	300	386	0	0.98	87.13
SCPb3	300	3 000	300	407	0	0.99	86.43
SCPb4	300	3 000	300	417	0	0.85	86.10
SCPb5	300	3 000	300	388	0	0.95	87.06
SCPc1	400	4 000	400	477	0	1.11	88.07
SCPc2	400	4 000	386	498	1	1.12	87.98
SCPc3	400	4 000	400	559	0	1.31	86.02
SCPc4	400	4 000	400	518	0	1.11	87.05
SCPc5	400	4 000	385	488	1	1.35	88.25
SCPd1	400	4 000	400	514	0	1.67	87.15
SCPd2	400	4 000	400	549	0	1.67	86.27
SCPd3	400	4 000	400	583	0	1.79	85.42
SCPd4	400	4 000	400	521	0	1.65	86.97
SCPd5	400	4 000	400	490	0	1.70	87.75

3.3　测试结果分析

由表 2 中可以看出,45 个测试用例的约简所运行时间非常少,其平均约简规模比为 86.88%,最大约简规模甚至超过 96%。可见本文算法的约简效率比较好,得到的数据优化度较原始数据有了很大的提高,达到了设计该算法的预期效果。

4　与其他算法的结合应用

本文算法作为对 SCP 数据的约简算法,与其他算法相结合来解决 SCP 问题更能体现出本算法在约简上的优点。下面用 Beasley 提出的 45 个测试用例对结合了约简算法的贪心算法和遗传算法进行实际测试,从而阐述本算法在提高传统算法

和智能算法解决 SCP 问题效率上的作用。

4.1　与贪心算法的结合

贪心算法原理为每次寻找一个相对代价最小的列加入解中,并去除该列所覆盖的行,直至所有行都被覆盖。将贪心算法与本文算法的结合称为二阶段贪心算法,两个算法阶段性运行,先运行约简算法,再对约简后的数据运行贪心算法。

表 3 给出了运行贪心算法和二阶段贪心算法的代价及运行时间。为方便比较,定义约简时间比为:

$$1 - T_{g'}/T_g \tag{15}$$

表 3　贪心算法约简前后运行时间比较表

测试用例	最优解	贪心算法		二阶段贪心算法		约简时间比/%
		代价	Tg/s	代价	Tg'/s	
SCP41	429	463	1.27	458	0.22	82.67
SCP42	512	582	1.28	582	0.35	72.65
SCP43	516	598	1.34	598	0.38	71.64
SCP44	494	548	1.28	548	0.33	74.21
SCP45	512	577	1.28	561	0.30	76.56
SCP46	560	615	1.21	618	0.40	66.94
SCP47	430	476	1.17	480	0.30	74.35
SCP48	492	533	1.17	533	0.33	71.79
SCP49	641	747	1.31	747	0.41	68.70
SCP410	514	556	1.28	557	0.29	77.34
SCP51	253	289	2.52	289	0.51	79.76
SCP52	302	348	2.46	348	0.57	76.82
SCP53	226	246	2.36	246	0.49	79.23
SCP54	242	265	2.49	265	0.53	78.71
SCP55	211	236	2.45	234	0.39	84.08
SCP56	213	251	2.47	251	0.46	81.37
SCP57	293	26	2.5	326	0.55	78.00
SCP58	288	323	2.59	323	0.57	77.99
SCP59	279	307	2.39	301	0.48	79.91
SCP510	265	293	2.50	290	0.51	79.60
SCP61	138	159	0.73	159	0.28	61.64
SCP62	146	170	0.69	170	0.31	55.07
SCP63	145	161	0.70	161	0.32	54.28
SCP64	131	149	0.74	149	0.32	56.75
SCP65	161	196	0.74	196	0.35	52.70
SCPa1	253	288	4.50	288	1.15	74.44
SCPa2	252	284	4.56	284	1.22	73.24
SCPa3	232	270	4.66	270	1.17	74.89
SCPa4	234	278	4.54	278	1.13	75.11
SCPa5	236	271	4.60	271	1.24	73.04
SCPb1	69	77	2.13	77	1.11	47.88
SCPb2	76	86	2.19	86	1.20	45.20
SCPb3	80	89	2.24	89	1.23	45.08
SCPb4	79	89	2.34	89	1.11	52.56
SCPb5	72	78	2.13	78	1.15	46.01
SCPc1	227	258	7.34	258	1.95	73.43
SCPc2	219	258	7.28	259	1.97	72.93
SCPc3	243	276	6.82	276	2.21	67.59
SCPc4	219	257	6.98	257	1.98	71.63
SCPc5	215	233	6.81	233	2.15	68.42
SCPd1	60	74	3.35	74	2.00	40.29
SCPd2	66	74	3.30	74	2.02	38.78
SCPd3	72	83	3.37	83	2.17	35.60
SCPd4	62	71	3.58	71	2.01	43.85
SCPd5	61	69	3.35	69	2.01	40.00

其中:T_g 为贪心算法的运行时间,$T_{g'}$ 为二阶段贪心算法的运行时间。从表中统计得出:a)对每个测试用例,二阶段贪心算法得到的解的代价与谈心算法得到的解的代价相近;b)所有测试用例的平均约简时间比为 66.06%;c)结合表 2 发现:对于同一测试用例,其约简规模比越大,约简时间比越大。

由于贪心算法的设计思想,使得某些约简后得到的解并没有比约简前得到的解要更优,但很明显,约简后算法所运行的时间大大缩短了。

4.2　与遗传算法结合

遗传算法是直接在原始数据中搜索解的一种智能算法,同样地,二阶段遗传算法是指先运行约简算法,然后再运行遗传算法,在约简后的数据中搜索解。表 4 给出了遗传算法和二阶段遗传算法运行 50 代所需时间及代价,同时给出了 100 代以内两算法所达到的最好解。表中括号内数字表示得到这个解所运行的代数。

表 4　遗传算法约简前后运行时间表

测试用例	最优解	遗传算法(50)		二阶段遗传算法(50)		约简时间比/%	遗传算法(≤100)		二阶段遗传算法(≤100)	
		C	T_g	C	$T_{g'}$		C	T	C	T
SCP41	429	432	288.32	429	26.91	90.66	431(60)	352.40	429(50)	26.91
SCP42	512	521	287.30	512	78.3	72.74	521(50)	287.30	512(50)	78.30
SCP43	516	526	287.58	520	81.97	71.49	520(60)	355.49	516(60)	94.84
SCP44	494	500	288.08	504	71.05	75.33	497(100)	575.89	496(80)	112.07
SCP45	512	518	286.88	512	51.84	81.92	516(100)	573.33	512(50)	51.84
SCP46	560	569	287.29	560	71.32	75.17	565(100)	575.28	560(50)	71.32
SCP47	430	443	286.42	432	59.46	79.24	431(60)	357.64	430(60)	69.09
SCP48	492	502	286.61	497	74.90	73.86	501(80)	466.62	492(60)	84.07
SCP49	641	662	288.87	641	86.16	70.17	655(80)	477.05	641(50)	86.16
SCP410	514	543	288.20	517	44.32	84.62	517(100)	578.00	517(50)	51.72
SCP51	253	274	546.13	255	82.55	84.88	258(60)	672.66	253(70)	111.78
SCP52	302	313	546.67	308	89.72	83.58	313(50)	546.67	302(60)	104.00
SCP53	226	229	544.31	228	80.78	85.15	229(50)	544.31	228(50)	93.69
SCP54	242	247	545.66	243	79.74	85.38	244(100)	1093.94	243(50)	92.49
SCP55	211	212	544.17	212	35.48	93.47	212(50)	544.17	212(50)	42.79
SCP56	213	219	543.89	213	63.25	88.37	213(60)	664.69	213(50)	63.25
SCP57	293	308	545.84	293	66.48	87.82	305(60)	667.11	293(50)	66.48
SCP58	288	314	545.92	288	85.33	84.36	288(80)	867.45	288(50)	85.33
SCP59	279	283	544.00	280	61.11	88.76	281(60)	665.13	280(50)	61.11
SCP510	265	275	546.06	269	64.78	88.13	272(80)	869.21	265(100)	262.28
SCP61	138	144	279.31	144	67.31	75.90	144(50)	279.31	138(100)	135.79
SCP62	146	154	279.27	146	74.55	73.30	150(80)	444.78	146(50)	74.55
SCP63	145	148	278.52	148	72.16	74.09	145(100)	558.52	145(60)	86.60
SCP64	131	133	279.95	131	64.15	77.08	131(100)	560.08	131(50)	64.15
SCP65	161	177	280.45	163	68.02	75.74	162(80)	444.91	161(60)	82.82
SCPa1	253	260	1201.41	253	175.28	85.41	257(100)	2397.79	253(50)	175.28
SCPa2	252	270	1204.72	265	187.26	84.45	262(100)	2398.40	253(60)	226.55
SCPa3	232	240	1202.08	234	185.14	84.58	238(80)	1921.01	233(80)	185.14
SCPa4	234	255	1199.26	235	182.25	84.80	238(80)	1911.60	234(80)	288.52
SCPa5	236	242	1200.68	237	184.18	84.66	239(100)	2398.21	237(50)	184.18
SCPb1	69	70	1189.28	69	170.42	85.67	70(50)	1189.28	69(60)	204.59
SCPb2	76	84	1193.16	80	179.59	84.94	82(100)	2377.44	78(60)	215.20
SCPb3	80	82	1191.17	80	186.65	84.33	81(60)	1452.17	80(50)	186.65
SCPb4	79	84	1190.18	83	190.57	83.98	81(60)	1451.13	79(60)	229.37
SCPb5	72	73	1189.79	72	180.80	84.80	72(60)	1449.95	72(50)	180.80
SCPc1	227	234	2122.33	232	307.37	85.51	231(100)	4221.99	229(60)	370.47
SCPc2	219	227	2116.54	223	307.76	85.45	227(50)	2116.54	219(60)	369.51
SCPc3	243	263	2120.49	248	347.51	83.61	253(80)	3351.55	245(70)	486.05
SCPc4	219	224	2113.83	224	335.07	84.14	224(50)	2113.83	219(60)	392.08
SCPc5	215	217	2118.27	219	302.99	85.69	217(50)	2118.27	215(60)	363.48
SCPd1	60	63	2103.60	62	307.55	85.37	63(50)	2103.60	60(100)	611.99
SCPd2	66	69	2105.04	68	321.28	84.73	69(50)	2105.04	67(80)	512.69
SCPd3	72	77	2105.32	73	338.39	83.92	77(50)	2105.32	73(50)	338.39
SCPd4	62	65	2110.49	63	310.16	85.30	63(100)	4197.50	62(100)	616.10
SCPd5	61	69	2101.85	64	291.02	86.15	64(60)	2559.56	61(100)	464.76

从表中可以看出:

a)在运行时间上,运行相同代数,二阶段遗传算法所需的时间都远小于遗传算。b) 45 个测试用例平均约简时间比为 82.64%。c)50 代以内纯遗传算法均没有达到最优解,而二阶段遗传算法有 13 组测试用例达到最优解。d)100 代以内遗传算法达到最优解的测试用例有 5 组,二阶段遗传算法有 32 组最优解且得到最优解的运行时间远小于遗传算法所得。除此之外,其余未达到最优解的测试用例所得的解也非常接近于最优解。e)相比于传统算法而言,该算法对于智能算法的辅助约简效果要更加明显。

5　结束语

本文通过对 45 个测试用例的约简,并结合贪心算法与遗传算法的应用,得出结论:与单纯的贪心算法和遗传算法相比,二阶段贪心算法和二阶段遗传算法能够在更少的时间内得到更优的解,表明该约简方法和约简算法可以有效提高传统算法和智能算法解决大规模集合覆盖问题的效率。

参考文献:

[1] MARSTEN R E, MULLER M R, KILLION C L. Crew planning at flying tiger: a successful application of integer programming[J]. Management Science, 1979, 25(12): 1175-1183.

[2] HOFFMAN K L, PADBERG M. Solving airline crew scheduling problems by branch-and-cut[J]. Management Science, 1993, 39(6): 657-682.

[3] RUBIN J. A technique for the solution of massive set covering problems with application to airline crew scheduling[J]. Transportation Science, 1973, 7(1): 34-48.

[4] GARFINKEL R. Integer programming[M]. [S. l.]: Wiley Intersecience, 1972.

[5] BREUER M A. Simplification of the covering problem with application to boolean expressions[J]. Journal for the Association of Computing Machinery, 1970, 17(1):166-181.

[6] FREEMAN D R, JUCKER J V. The line balancing problem[J]. Journal of Industrial Engineering, 1967, 18(6): 361-364.

[7] FORSTER B A, RYAN D M. An integer programming approach to the vehicle scheduling problem[J]. Operational Research Quarterly, 1976, 27(2): 367-384.

[8] GARFINKEL R G, R, TRIPAHI A, YIN F. Design of a shopped and recommender system for bundle purchases[J]. Decision Support Systems, 2006, 42(3): 1974-1986.

[9] DAY R H. On optimal extracting from a multiple file data storage system: an application of integer programming[J]. Operations Research, 1965, 13(3): 482-494.

[10] GAREY M R, JOHNSON D S. Computers and intractability: a guide to the theory of NP-completeness[M]. San Francisco: W. H. Freeman, 1979.

[11] FISHER, M, KEDIA P. Optimal solution of set covering/partitioning problems using dual heuristics[J]. Management Science, 1990, 36(6): 674-688.

[12] CAPRARA A, FISCHETTI M, TOTH P. Algorithms for the set covering problem[J]. Annals of Operations Research, 2000, 98(1-4): 353-371.

[13] CHVATAL V. A greedy heuristic for the set-covering problem[J]. Mathematics of Operations Research, 1979, 4(3): 233-235.

[14] AICKELIN U. An indirect genetic algorithm for set covering problems[J]. Journal of the Operational Research Society, 2002, 53(10):1118-1126.

[15] BEASLEY J E, CHU P C. A genetic algorithm for the set covering problem[J]. European Journal of Operational Research, 1996, 94(2):392-404.

[16] SOLAR M, PARADA V, URRUTIA R. A parallel genetic algorithm to solve the set covering problem[J]. Computers & Operations Research, 2002, 29(9): 1221-1235.

[17] JACOBS L, BRUSCO M. Note: a local-search heuristic for large set covering problems[J]. Naval Research Logistics, 1995, 42(7): 1129-1140.

[18] LESSING L, DUMITRESCU I, STÜTZLE T. A comparison between ACO algorithms for the set covering problem[J]. Lecture Notes in Computer Science, 2004, 3172: 1-12.

[19] BEASLEY J E. A lagrangian heuristic for set covering problems[J]. Naval Research Logistics, 1990, 37(1):151-164.

[20] CERIA S, NOBILI P, SASSANO A. A lagrangian-based heuristic for large-scale set covering problems[J]. Mathematical Programming, 1998, 81(2): 215-228.

[21] OHLSSON M, PETERSON C, SE DERBERG B. An efficient mean field approach to the set covering problem[J]. European Journal of Operational Research, 2001, 133(3): 583-595.

[22] BEASLEY J E. An algorithm for set covering problems[J]. European Journal of Operational Research, 1987, 31(1): 85-93.

(上接第 3299 页)

[25] LANG J C, WIDJAJA T. Optimizing the supplier selection and service portfolio of a SOA service integrator[C]//Proc of the 41st Hawaii International Conference on System Sciences. 2008:1-10.

[26] KIM D, LEE S, ABRAHAM A, *et al*. A dynamic priority allocation scheme of message for differentiated Web services satisfying service level agreement[J]. Journal of Digital Information Management, 2006, 4(1):26-31.

[27] TANG Q C, CHENG H K. Optimal location and pricing of Web services intermediary[J]. Decision Support Systems, 2005, 40: 129-141.

[28] CHENG H K, TANG Q C, ZHAO J L. Web services and service-oriented application provisioning: an analytical study of application service strategies[J]. IEEE Trans on Engineering Management, 2006, 53(4):520-533.

[29] AGARWAL S, LAMPARTER S. Making Web Services tradalde a poliay based approach for specifying perferences on Web Service properties[J]. Web Semantics: Service and Ageats on World Wide Web, 2009(7):11-20.

[30] 邵凌霜,周立,赵俊峰. 一种 Web service 的服务质量预测方法[J]. 软件学报, 2009, 20(8):2062-2073.

[31] CASOLA V, MANCINI E P, MAZZOCCA N, *et al*. Self-optimization of secure Web Services[J]. Computer Communications, 2008, 31(18):4312-4323.